Probabilistic and dynamic optimization of job partitioning on a grid infrastructure

Tristan Glatard, Johan Montagnat, Xavier Pennec

Probabilistic and Dynamic Optimization of Job Partitioning on a Grid Infrastructure

Tristan Glatard, Johan Montagnat, Xavier Pennec

Laboratoire Informatique Signaux et Systèmes (I3S) – CNRS, France
INRIA Sophia-Antipolis, France

Abstract

Production grids have a potential for parallel execution of a very large number of tasks but also introduce a high overhead that significantly impacts the execution of short tasks. In this work, we present a strategy to optimize the partitioning of jobs on a grid infrastructure. This method takes into account the variability and the difficulty to model a multi-user large-scale environment used for production. It is based on probabilistic estimations of the grid overhead. We first study analytically modeled environments and then we show results on a real grid infrastructure. We demonstrate that this method leads to a significant time speed-up and to a substantial saving of the number of submitted tasks with respect to a blind maximal partitioning strategy.

Keywords : Grid Computing – Models and Tools – Heterogeneous Systems – Parallel Systems – Distributed Systems

1. Introduction

The problem of sharing independent tasks over a set of a known amount of computing resources connected through a reliable and high performance network has been tremendously studied in the field of parallel computing [2, 5]. In this paper we are addressing a similar problem arising when considering execution of a set of independent tasks on a computing grid. A grid is understood here as a set of individual computers, connected through a not-so-reliable wide area network and which topology can dynamically change over time.

When using grid resources, a user get access to a considerable and potentially infinite amount of distributed computing power. Ideally, a user would split his computing jobs in as many independent tasks as computing resources available. In practice though, usage of grid resources introduces an overhead to the computing time due to the time needed to reach remote resources, the scheduler working time, the queuing time, and so on. Moreover, the infrastructure is actually of course limited and its performance is likely to decrease if the total number of submitted tasks exceeds a certain limit. Therefore, optimal computation time is usually not obtained by maximum splitting: a trade-off has to be found between splitting and grid overhead that tend to grow with the number of tasks to deal with.

In the context of grid computing, the problem of tasks splitting is impacted by several new parameters: the grid resources are highly volatile, the topology of the grid network is ever changing, and the status of the whole grid is never fully known at a given instant. In the rest of the paper, we study the underlying hypotheses and we survey the literature. Since a quick overlook of the literature does not satisfy us, we propose a probabilistic approach to estimate the grid computation overhead. We demonstrate the relevance of our model through an analytical study and we show results on a real full scale grid used for production.

1.1. Hypothesis and context

Grid is a rather overused term covering a very broad area. Among the different approaches on distributed systems that can be qualified as grids and that are currently being investigated for research, we are focusing on early grid systems originating from cluster computing that are being put in production today [4, 3, 9, 6, 15]. In such a system, the grid is mostly an aggregation of clusters, each using a classical batch system to handle its local computing load, and a *middleware* software layer offering an interface to these potentially heterogeneous batch systems and resources, with a single sign-on entry point for users. Such a grid is merely a super-batch system capable of handling tremendous amounts of computations, and particularly efficient in processing independent and large grain computations.

Batch computing is restrictive as compared to modern, highly dynamic systems. Yet, these systems are robust and able to deal with legacy code without requiring any rewriting nor instrumentation. A grid deployed over a wide area network proves to have highly volatile resources: the prob-

ability for hardware failure is growing with the amount of resources and the network instability can temporarily disconnect full fragments out of the infrastructure. Moreover, resources may appear or disappear depending on the needs for maintenance operations, addition to the infrastructure, etc. The full grid topology and status cannot be known at a given instant: grid information providers [17] rely on past and possibly outdated data on the resources. As production grids are multi-users (and even multi-community) systems, they are continuously loaded. The evolution of load put on grid resources is yet another hardly predictable parameter. Production systems are only accessible for end users by their user interface giving minimal access to their internal mechanisms. On production grids, we have no access to the core middleware and we can mainly submit computing tasks, monitor their evolution, and collect results, to obtain information on the grid status.

In this paper, we are conducting experiments on the EGEE infrastructure which corresponds to the description given above (see section 5.1). The best suited application for such an infrastructure are so called embarrassingly parallel applications: computing tasks with a very high level of inherent parallelism and no much synchronization constraints. High Energy Physics computing is famous for that kind of applications, and other example in the field of medical imaging are described in the literature [11, 14]. In particular, the french project AGIR aims at deploying medical image based applications on grids [7].

1.2. Formalization

Problem description. On real large scale grid infrastructures, the overhead due to submission, scheduling, etc, can be of a few minutes. Thus, the submission of short-running tasks is likely to slow-down the global application. Tasks should be grouped into larger sets to reduce the impact of this penalty. Moreover, optimizing the tasks' granularity reduces the total number of tasks submitted to the infrastructure with respect to the default maximal partitioning strategy. Hence, a potential improvement of the grid performance can be obtained if every user adopts such a strategy. Consequently, a trade-off has to be found between submitting a high number of short tasks, which maximizes parallelism but may leads the grid overhead to prevail on the running time, and submitting a small number of longer tasks.

In this context, our ultimate goal is to propose an optimization strategy for tuning the granularity of the tasks submitted to the grid, given a fixed job to execute. This strategy would aims at :

1. Lowering the total execution time of a job (user's point of view) ;
2. Reducing the total number of tasks submitted for a given job (infrastructure's point of view).

Problem modeling. Given a total job corresponding to a known CPU time W supposed to be divisible into any number n of independent tasks and a grid infrastructure introducing an overhead G corresponding to the submission, scheduling and queuing time of the tasks, we consider that the execution of the whole task is completed when *all the tasks* are completed. We also make the hypothesis that a task will be affected to a single processor, so that the number n of submitted tasks strictly corresponds to the number of processors involved in the execution. Thus, the goal is to minimize the total execution time H defined as:

$$H = max_{i\in[1,n]}\left(G + \frac{W}{n}\right)$$

If we assume G to be a fixed value, the solution is straightforward and n has to be as high as possible. However, this assumption is not realistic in most cases, due to the infrastructure's nature. A more realistic view is to assume G to be dependent both on i and time. In the next section, we review works related to this problem.

2. Related work

A complete comparative study of allocation strategies is presented in [8], where task execution times are modeled as random variables with known mean and variance. The author demonstrates the need for a dynamic allocation strategy and he points out that it is associated with an overhead which is assumed to be fixed in his study. He considers a system of initially idle processors and notices that there is a trade-off between overhead and idle time. Under those assumptions, he compares several allocations strategies by presenting simulation results. Our problem cannot be entirely addressed by such methods because (i) allocation strategies consist in optimizing the size of the batches allocated to the processors whereas we are trying to optimize the total number of processors to use, given that every processor will be given a W/n CPU time, (ii) we cannot assume the overhead G to be fixed as it varies along time in our case, and (iii) on a production infrastructure the processors are never idle. This leads to a queuing time that we take into account in the G random variable.

Scheduling techniques are largely studied in the literature. A detailed review of heuristics and a study the impact of performance estimation on the scheduling strategy is presented in [1]. The authors assume that the scheduler has knowledge of the current topology of the grid, the number and location of copies of all input files and the list of computations and file transfers currently underway or already completed. This kind of solution is hardly usable

in our case because we do not assume any a priori knowledge about the infrastructure concerning computations in progress or the current grid topology. In [13] a scheduling method taking into account the stochastic nature of the time to compute one unit of data on a distant processor, supposing that distributions are Gaussian. In particular, the authors notice that penalizing highly variable processors leads to a significant speed-up. Even if the variability of resources has to be taken into consideration (see section 3.1), this approach cannot totally suit with our problem because (i) the distribution of G is not assumed to be Gaussian (and is actually not, as described in section 4), and (ii) the distribution of G has to be dynamically estimated in the multi-users infrastructure we consider. In [12], the author presents decision rules for sequential resource allocation based on dynamic programming. They consider the problem consisting in allocating machines to sequentially arriving tasks. Even if this kind of solution would constitute an interesting perspective, our problem seems not to be treatable by dynamic programming methods because we here consider a set of independent tasks being all submitted in parallel, at the same time, to the infrastructure.

Other works address the task granularity issue, noticing that there is an optimal number of processors to determine to minimize the total execution time, taking into account both computation time and communication time. In [16], they use heuristics to determine a close to optimal configuration, in which tasks are assigned to specific processors to reduce communication overhead induced by routing and contention. Even if it provides good results in their scope, their solution is strictly deterministic and models the communication function linearly in the number of processors, which cannot properly describe the overhead G we need to consider. In [11], the authors determine the optimal number of tasks to submit by determining an analytical model of the overhead of the grid submission and queuing system in a batch architecture. Such an analytical model is very hard to determine in a complex dynamic multi-users grid infrastructure.

Works such as [10] and inside references propose performance analysis methods for task scheduling into embedded systems, considering probabilistic models of task execution times. In this work, the authors model task execution by a generalized continuous probability distribution and propose a method not restricted to any specific scheduling policy. They consider both execution time and memory aspects. Their method is based on the construction of an underlying stochastic process and its analysis. Even if this approach is entirely probabilistic and makes no assumption on the nature of the probability function of the execution time, which well suits with our hypotheses, they assume all the tasks to be executed concurrently on a single processor.

As a conclusion, the above methods does not seem to completely match our hypotheses. Therefore, we propose a dedicated model in the rest of the paper. We adopted a probabilistic approach to cope with the variation of the overhead G among the tasks. This approach is detailed in the next section. We address the problem of the dependency of G along time with a dedicated infrastructure monitoring system that is presented in section 4. Section 5 shows results and an evaluation of the proposed model on a production grid.

3. A probabilistic model

In this section, we investigate the problem described in 1.2 considering that G is a random variable. If we assume the probabilistic density function (p.d.f) of the random variable G to be $f_G(t)$, then the p.d.f of H will be $f_H(t)$, such as:

$$\begin{aligned} F_H(t) &= P(H < t) = \prod_{i=1}^{n} P\left(G + \frac{W}{n} < t\right) \\ &= P\left(G < t - \frac{W}{n}\right)^n = F_G\left(t - \frac{W}{n}\right)^n \\ \text{Then } f_H(t) &= \frac{dF}{dt} = n.f_G\left(t - \frac{W}{n}\right).F_G\left(t - \frac{W}{n}\right)^{n-1} \end{aligned}$$

The problem can then be formulated as a minimization with respect to n of the expectation E_H of the random variable H:

$$\begin{aligned} E_H(n) &= \int_{\mathbb{R}} t.f_H(t)dt \\ &= \int_{\mathbb{R}} t.n.f_G\left(t - \frac{W}{n}\right)F_G\left(t - \frac{W}{n}\right)^{n-1} dt \\ &= \int_{\mathbb{R}} n.\left(t + \frac{W}{n}\right).f_G(t).F_G(t)^{n-1}dt \\ &= \int_{\mathbb{R}} n.t.f_G(t).F_G(t)^{n-1}dt + \frac{W}{n} \end{aligned}$$

3.1. Application to synthetic distributions

In this section, we investigate the problem analytically considering synthetic distributions for G, in order to demonstrate the relevance of the method in a controlled environment.

If we for example assume G to be uniformly distributed between a minimum value a and a maximum value b, then an explicit solution can be provided: indeed, we then have:

$$f_G(t) = \begin{cases} \frac{1}{b-a} \text{ if } t \in [a,b] \\ 0 \text{ else} \end{cases}$$

and

$$F_G(t) = \begin{cases} 0 \text{ if } t < a \\ \frac{t-a}{b-a} \text{ if } t \in [a,b] \\ 0 \text{ if } t > b \end{cases}$$

$$E_H(n) = \int_a^b n.t.\frac{1}{b-a}.\left(\frac{t-a}{b-a}\right)^{n-1} dt + \frac{W}{n}$$
$$E_H(n) = \frac{(n+1).W + b.n^2 + a.n}{n.(n+1)}$$

This result is coherent as $E_H(1) = W + \frac{a+b}{2}$: the execution time on a single CPU is W and the execution suffers from a $\frac{a+b}{2}$ penalty that is the mean overhead introduced by the infrastructure. Moreover, $\lim(E_H(n))_{n\to+\infty} = b$: with an infinite amount of resources, it corresponds to the worst possible overhead introduced by the grid (b) and to the best computation time (0). Indeed, as the number of submitted tasks increases, the probability for one of the tasks to suffer from a high overhead increases. Finally, $\lim(E_H(n))_{n\to 0} = +\infty$: the limit of E_H towards zero corresponds to the execution of the task on zero machine. In this case, the execution time of course tends to infinity.

The next step is the minimization of the expectation of H. Let us consider its derivative with respect to n:

$$\frac{dE}{dn} = -\frac{n^2.W + 2nW + W - b.n^2 + a.n^2}{n^2.(n+1)^2}$$

$$\text{If } W \neq b-a \quad \text{and} \quad \frac{dE}{dn} = 0$$

$$\text{Then} \quad \begin{cases} n_1 = -\frac{\sqrt{(b-a)W}+W}{W-(b-a)} \\ or \\ n_2 = \frac{\sqrt{(b-a)W}-W}{W-(b-a)} \end{cases}$$

n_1 is positive if $(b-a) > W$ and negative otherwise whereas n_2 is always negative. Given that n has to be positive, there is thus a unique optimal number of tasks n_{opt} minimizing $E_H(n)$ if $(b-a) > W$ and we have: $n_{opt} = -\frac{\sqrt{(b-a)W}+W}{W-(b-a)}$. Such a configuration is represented on the left graph of figure 1 where we plotted $E_H(n)$ for a uniform distribution with a = 200s, b = 4000s and W = 2000s. On the other hand, if $(b-a) < W$ then $\frac{dE}{dn} < 0$ so that E_H is strictly decreasing and the optimal number of tasks corresponds to the maximal one. Such a configuration is represented on the right graph of figure 1 where we plotted $E_H(n)$ for a uniform distribution with a = 700s, b = 1500s and W = 2000s. If $W = b-a$, then $\frac{dE}{dn} = -\frac{2nW+W}{n^2.(n+1)^2}$: it thus has no positive root and here again, the optimal number of tasks corresponds to the maximal one.

We thus can conclude from this particular example that the relative variability $V = \frac{b-a}{W}$ of the grid overhead G plays a strong role into the optimization procedure: whatever the actual mean of G is, if V is low enough, then looking for an optimal job partitioning does not make sense. Indeed, in that case, G can be seen as a fixed value with respect to W and the problem is straightforward, as we explained in 1.2.

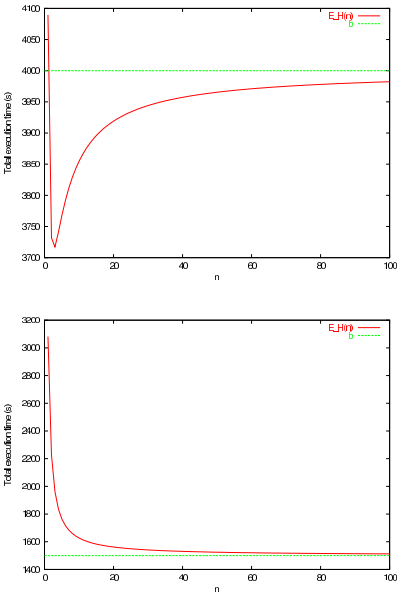

Figure 1. Representation of $E_H(n)$ for a uniform distribution with a = 200s, b = 4000s and W = 2000s (up) and a = 700s, b = 1500s and W = 2000s (down)

If we now suppose the distribution of G to be Gaussian, with mean μ and standard deviation σ, then:

$$f_G(t) = \frac{1}{\sqrt{2.\pi}.\sigma}.exp\left(-\frac{(t-\mu)^2}{2.\sigma^2}\right)$$
$$\text{and} \quad F_G(t) = \frac{1}{\sqrt{2.\pi}.\sigma}\int_{-\infty}^{t} exp\left(-\frac{(u-\mu)^2}{2.\sigma^2}\right)du$$

$$E_H(n) = \int_{\mathbb{R}} n.t.\frac{1}{\sqrt{2.\pi}.\sigma}.exp\left(-\frac{(t-\mu)^2}{2.\sigma^2}\right). \left(\frac{1}{\sqrt{2.\pi}.\sigma}\int_{-\infty}^{t} exp\left(-\frac{(u-\mu)^2}{2.\sigma^2}\right)du\right)^{n-1}.dt + \frac{W}{n}$$

In this case, the relative variability V of the overhead G is denoted by $V = \frac{\sigma}{W}$. Minimizing $E_H(n)$ is hardly analytically feasible but we can estimate a minimum numerically. For example, if we consider μ = 600s and σ = 300s, figure 2 displays the evolution of $E_H(n)$ with respect to n for different values of V ranging from 0.015 to 0.6. We can see on those figures that the higher the relative variability is, the deeper the minimum of $E_H(n)$ is. One can here again conclude that the optimization procedure is particularly suited for environments with a high variability with respect to W.

Applying the model on synthetic distributions showed that it seems coherent and that it is particularly adapted to

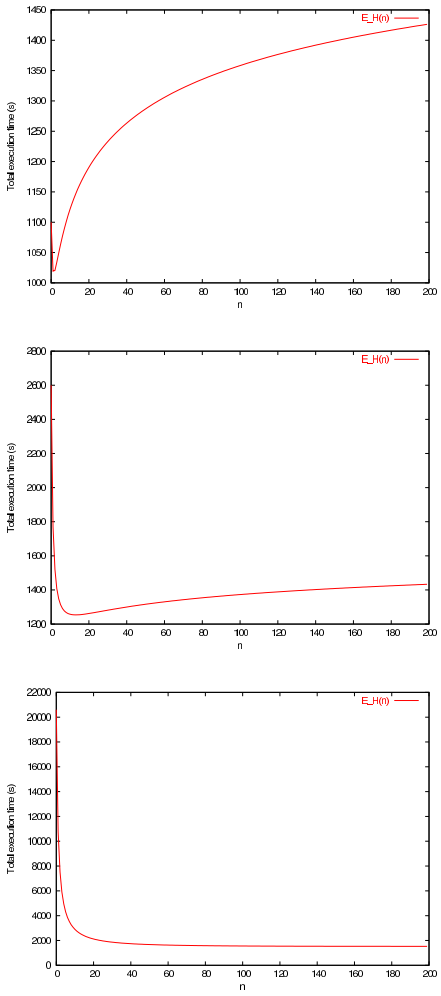

Figure 2. $E_H(n)$ **for a Gaussian distribution with** $\sigma = 300$**s and** $\mu = 600$**s. From top to bottom:** $V = 0.6$**,** $V = 0.15$ **and** $V = 0.015$

highly variable environments. But as stated in the introduction, we cannot assume the p.d.f of G to be known. Therefore, we present in the next section the method we used to estimate it from measures.

4. Experimental distributions

This section describes the method we used to estimate the p.d.f of G from measures.

4.1. Measuring task times

Our optimization method is based on the evaluation of the p.d.f of the infrastructure's latency G. Thus, our primary goal was to determine a robust procedure to measure it. Ideally a grid infrastructure should provide this measure from all the tasks submitted by the users. However from our user's point of view, we cannot access the statistics concerning all the tasks submitted to the infrastructure. Thus, the experimental method we adopted was to submit waves of dedicated "ping" tasks to the infrastructure. Those tasks do not process anything and we use them as probes to measure the grid latency, by monitoring their submission, scheduling and queuing times.

The main problem it raises is the fact that the status of the infrastructure may be disrupted by such a measure. Indeed, submitting waves of measure tasks would cause an additional load on the infrastructure, leading to inconsistent measures. To face this problem, we initially submit a limited set of "ping" tasks and then instantaneously submit a new one each time a "ping" task completed, so that the total number of measure tasks running on the infrastructure is constant, leading to a fixed perturbation.

Even if a grid potentially provides an infinite number of resources, and thus allows a theoretical infinite number of task submission, a real infrastructure is actually limited by the maximum number of simultaneous connections from the submission entity and the maximum number of tasks on the scheduler. We empirically tuned the number of "ping" tasks as a trade-off between the accuracy of the measure and the induced overhead. On the target grid infrastructure, we used 50 measure tasks.

It is true that this kind of method is quite unfair because it introduces a significant overload on the infrastructure. But ultimately, the middleware should provide to the users such statistics computed from all the submitted tasks so that the method would not be invasive.

4.2. Timeouts

On a real large scale multi-users grid infrastructure task "losses" may occur because of *e.g.* overfull waiting queues, execution failures on distant heterogeneous machines, network problems and so on. Therefore, setting a timeout to tasks is required to avoid unreasonable waiting times. Taking into account timed-out tasks into the optimization procedure would require to propose a fault-tolerant system handling task resubmissions and so on. Even if we know that it is a very important problem, this will be part of our future work. In this present work, we focus on the validation of the global principle and we thus decided to neglect timed-out tasks, both in the measure scope and in the validation study.

Setting the timeout of tasks to get consistent measures is not straightforward. Indeed, when a measure comes back from the infrastructure, it describes the infrastructure status at the measure's submission instant. Thus, the timeout has

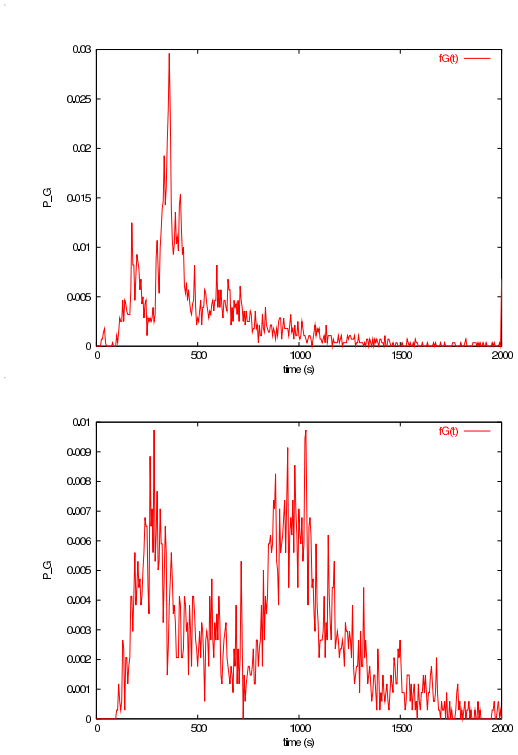

Figure 3. Examples of p.d.f of G

to be inferior to the duration while we could consider that the infrastructure's status does not vary. On the other hand, the timeout has to be large enough not to discard too many tasks. For our experiments, we fixed the timeout of tasks to the total CPU time value W of the task, so that timed-out tasks are the ones which would lead to a slowing down of the task by the grid execution.

4.3. Estimating and minimizing the probabilistic density function of G

Once we have latency measures, the next step is to determine the p.d.f of the infrastructure's latency G. We did that by considering the 50 last ping measures and gathering them into 5 seconds bins. Obtaining the corresponding p.d.f is then straightforward.

To provide an idea of the overhead times, two sample examples of the p.d.f of G at a given instant are displayed on figure 3. As we can see on this figure, the p.d.f is likely to strongly vary along time. Moreover, we can notice that those distributions are clearly not Gaussian. Indeed, they both are neither mono-modal, nor symmetric with respect to their mean.

Once we estimated the p.d.f of G the computation and minimization of $E_H(n)$ is straightforward, according to section 3. We just computed $E_H(n)$ with n ranging from 1 to a maximum value corresponding to the maximum number of tasks submittable to the infrastructure from a single user interface.

	Min	Max	Avg	Median
δ (seconds)	10	960	258.94	215
$\delta_{normalized}$	0.04	12.64	2.1	1.16

Table 1. Errors between model and measures

5. Experiments and Results

5.1. Infrastructure

We evaluated the proposed model on the grid infrastructure provided by the EGEE European project [1]. The platform offered is a pool of thousands computing (standard PCs) and storage resources accessible through the LCG2 middleware [2]. The resources are assembled in computing centers, each of them running its internal batch scheduler. Tasks are submitted from a user interface to a central Resource Broker which distributes them to the available resources. This infrastructure strictly matches the hypothesis we did on section 1.1. Indeed, from an application point of view, this infrastructure behaves as a batch scheduler and we do not have any information on the resources.

5.2. Experiments

We made two experiments to evaluate our model on the EGEE infrastructure.

First, we evaluated the model capability to correctly predict the execution time of a set of tasks on the grid infrastructure. We submited and measured the total execution time of a job, having previously estimated this time with $E_H(n)$. The job is composed of 30 tasks, 67 seconds long each, thus leading to a total CPU time W of 2000 seconds.

Second, we quantified the benefit induced by the model (*optimal strategy*) compared to the naive strategy consisting in submitting a maximal number of tasks (*maximal strategy*). A total CPU time $W = 2000$s is submitted, on the one hand using the optimal number of tasks resulting from the minimization of $E_H(n)$, and on the other hand using a fixed number of 30 tasks (this corresponds to the maximum number of tasks we can submit concurrently on the infrastructure without hitting some performance loss). To avoid bias resulting from an evolution of the grid status between the two submission processes, we alternatively repeated the

[1] http://www.eu-egee.org
[2] http://lcg-web.cern.ch

	Min	Max	Avg
Expected	0	671	162.5
Measured	-775	1308	198.1

Table 2. Time speed-up (s) between maximal and optimal strategies

two strategies up to 88 times, on various day times (mornings, afternoons, nights) spread over a week and using 3 different Resource Brokers.

5.3. Results

Experiment 1: model vs measures. Table 1 shows on its upper line statistics concerning the difference δ in seconds between the model prediction and the effective measure. In order to quantify the accuracy of the model, we normalized this error with the predicted standard-deviation of the random variable H : $\delta_{normalized} = \frac{\delta}{\sigma_H}$. The table thus shows on its lower line the minimum, maximum, average and median ratios between the measured errors and the standard-deviation σ_H of the random variable H. One can notice that the median ratio is close to 1, that is to say that the measured error is close to the standard-deviation of H. We can thus conclude that the proposed model is effectively able to predict the execution time of a set of tasks on the grid infrastructure.

Experiment 2: optimal strategy vs maximal strategy. Two different conclusions can be made from this experiment.

Task saving: on the 88 experiments, the estimated optimal number of tasks *n* differed from the maximal one 37 times, that is to say in 42% of the experiments. The remaining 58% correspond to the experiments where the computed optimal *n* is 30. The total number of submitted tasks is 2580 for the maximal strategy and 1756 for the optimal one. One thus can see that the optimal strategy leads to a total saving of 824 tasks, representing 32% of the tasks submitted in the maximal strategy.

Time speed-up: table 2 shows statistics over the 88 executions on the differences (in seconds) between the maximal and the optimal strategies in cases where the computed optimal *n* differs from the maximal one. Negative values show that the maximal strategy was faster than the optimal one. One can notice that the average speed-up introduced by our optimization strategy is about 200s, which represents 10% of the total submitted CPU time W.

6. Conclusion and future work

In this work, we detailed a strategy to optimize the job partitioning on a real grid infrastructure. The method takes into account the dynamic and probabilistic nature of such an infrastructure by perpetually refreshing the p.d.f of the grid's overhead and minimizing the expectation of the total task time. We show experimental results demonstrating one the that (i) a significant speed-up and (ii) a substantial task saving can be obtained using this method.

However, parameters such as the data transfer time and the random nature of the computing power of the resources are not considered by our model. Including them into the partitioning strategy will be part of our future work. We also plan to consider timeouts and fault tolerance elements such as resubmissions in order to propose a more complete strategy for the optimization of job partitioning on a grid infrastructure.

7. Acknowledgement

This work is partially funded by the French research program "ACI-Masse de données", http://acimd.labri.fr/, AGIR project (http://www.aci-agir.org/).

References

[1] H. Casanova, G. Obertelli, F. Berman, and R. Wolski. The apples parameter sweep template : user-level middleware for the grid. In *ACM/IEEE conference on Supercomputing*, 2000.

[2] P. Chrétienne, C. E.G, J. Lenstra, and Z. Liu, editors. *Scheduling theory and its applications*. John Wiley and Sons, 1995.

[3] Condor, high throughput computing. http://www.cs.wisc.edu/condor/.

[4] European IST project of the FP6, Enabling Grids for E-sciencE, apr. 2004-mar. 2006. http://www.eu-egee.org/.

[5] D. Feitelson, L. Rudolph, and U. Schwiegelshohn. Parallel job scheduling – a status report. In *10th Workshop on Job Scheduling Strategies for Parallel Processing*, New-York, NY, June 2004.

[6] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *International Journal of Supercomputer Applications*, 11(2):115–128, 1997.

[7] C. Germain, V. Breton, P. Clarysse, Y. Gaudeau, T. Glatard, E. Jeannot, Y. Legré, C. Loomis, J. Montagnat, J.-M. Moureaux, A. Osorio, X. Pennec, and R. Texier. Grid-enabling medical image analysis. In *5-th IEEE/ACM International Symposium on Cluster Computing and the Grid – CCGrid – Bio-Grid workshop*, Cardiff, UK, May 2005. IEEE Press.

[8] T. Hagerup. Allocating independent tasks to parallel processors : An experimental study. *Journal of Parallel and Distributed Computing*, 47:185–197, 1997.

[9] Legion: worldwide virtual computer. http://legion.virginia.edu/.

[10] S. Manolache, P. Eles, and Z. Peng. Memory and time-efficient schedulability analysis of task sets with stochastic execution time. In *13th Euromicro Conference on Real-Time Systems*, Delft, The Netherlands, 2001.

[11] J. Montagnat, V. Breton, and I. E. Magnin. Medical image databases content-based queries partitioning on a grid. In *HealthGrid'04*, Clermont-Ferrand, France, Jan. 2003.

[12] L. Pronzato. Optimal and asymptotically optimal decision rules for sequential screening and resource allocation. *IEEE Transactions on Automatic Control*, 46(5):687–697, 2001.

[13] J. Schopf and F. Berman. Stochastic scheduling. In *Supercomputing*, Portland, USA, 1999.

[14] T. Tweed and S. Miguet. Distributed indexation of a mammographic database using the grid. international. In *Workshop on Grid Computing and e-Science, 17th Annual ACM International Conference on Supercomputing*, San Francisco, USA, June 2003.

[15] UNICORE: Uniform Interface to Computing Resources. http://unicore.sourceforge.net/.

[16] J. Weissman and X. Zhao. Scheduling parallel applications in distributed networks. *Journal of Cluster Computing*, 1998.

[17] R. Wolski. Dynamically forecasting network performance using the network weather service. *Cluster Computing*, 1(1):119–132, 1998.